A web-based system for collaborative electro-acoustic composition

**GIBSON, I. and DOVEY, M.**
School of Technology, Leeds Metropolitan University, Leeds, UK
e-Science Centre, University of Oxford, Oxford, Oxfordshire, UK

## ABSTRACT

This paper describes development of an online system for collaborative electro-acoustic music composition. Traditionally, software systems for music composition have been single user systems. As accessibility to the Internet has increased, some systems have been adapted so that musicians can produce collaborative work. However, many such systems are still based around the composition methods of the original single user systems. Various Service Oriented Architectures are emerging based on WebServices or GridServices and other Internet technologies for dynamically building systems out of distributed components. Many e-Science projects are using these architectures to build collaborative environments. The software described in this paper takes such concepts and tools from Service Oriented Architectures and e-Science, and applies them to develop software specifically for collaborative electro-acoustic composition on the Internet, allowing compositional tools and components to be published, discovered and used within a distributed environment. An objective is to explore and determine methods of composition appropriate for this environment.

## 1 BACKGROUND

In many cases, electronic sound synthesis techniques can produce a greater range of timbres than any one acoustic instrument. The composer-performer has access to a wide range of software and hardware tools for score writing, sound synthesis, sound editing and music arrangement.

Real-time control of synthesized sound is often achieved using a device compatible with the Musical Instrument Digital Interface (MIDI) protocol (1). However MIDI devices (such as keyboards) tend to have limited degrees of freedom (2). For this reason, many electro-acoustic composers choose to compose in non real-time using Music Computer Languages such as CSound (3) which allow the user to attain a higher resolution of control over parameters but at the expense of being more computationally intensive. Traditionally, these languages have taken the form of text-based interfaces whereby the computer is used to process a script. CSound stores ‘instrument

definitions' in an orchestra file and instructions for playing instruments in a score file.

## 2 SOUND SYNTHESIS TECHNIQUES

This section describes common techniques used by the composer-performer for synthesizing sounds.

### 2.1 Additive Synthesis

*Additive* or *Fourier synthesis* operates in the frequency domain. A waveform may be represented as a series of sinusoidal waveforms (*partials*). Fourier analysis may be used to analyse an audio signal for re-synthesis. Small sections of the original audio signal (*windows*) are taken and analysed in this way producing sets of partials with their amplitudes and phase. The size of the window must be larger than the longest pitch period to be analysed. To increase accuracy of the analysis stage, some fourier analysis programs allow interpolation between windows.

Re-synthesis becomes possible by generating sinusoidal oscillators which follow the amplitude envelopes generated in the analysis stage. When re-synthesising acoustic instruments in this way it is necessary to perform analysis over a range of frequency values. A note played on a lower register will typically have different partial envelopes to a note played on a higher register.

Fourier synthesis can also be used to generate new sounds and musical effects. A *morph* between two sounds may be achieved using interpolation between their respective envelopes, many examples of which can be found in the piece "Vox 5" by Trevor Wishart. Further interpolation may be applied within the time domain to resolve differences in attack or decay values between the two sounds. This method is also useful for creating a sound which has timbral qualities of both sounds.

The Digital Phase Vocoder (4) is an extension of fourier transform and has been used to control a synthesiser from a traditional instrument (5). It is used to map spectral characteristics of the acoustic instrument directly onto a synthesized sound using additive synthesis.

### 2.2 Subtractive Synthesis

Subtractive synthesis is based around the use of one or more of the following filters on a source sound: low-pass, high-pass, band-stop, and band-pass. These filter out parts of the sound spectrum, shaping the amplitude and phase of each spectral component. A low-pass filter permits frequencies below a cutoff frequency (fc) to pass. Above the frequency fc, the spectral components are reduced significantly in amplitude. The cutoff frequency is defined at that point at which the power transmitted drops to one half (-3 dB) of the maximum power transmitted.

A high-pass filter allows only signals above the cutoff frequency to pass unaffected. A band-pass filter rejects frequencies either side of two cutoff frequencies. These are specified by a centre frequency and a bandwidth. The band-stop filter is the inverse of a band-pass, attenuating frequencies within the specified bandwidth.

Because no new frequencies are introduced subtractive synthesis is particularly effective with complex sound sources. Sawtooth and triangle waves produce sounds rich in harmonic spectra but

due to their periodic nature they can sound predictable. Noise and pulse waves are also used due to their spectral richness. The pulse waveform has an amplitude which lasts just a brief period of time (determined by the pulse width). A narrow pulse produces a large amount of spectral energy in its higher frequencies relative to its low frequency components.

### 2.3 **Frequency Modulation**

**Frequency modulation (FM) occurs when the output of one oscillator** (the *modulator*) is used to modulate the frequency of another (the *carrier*). The output spectrum resulting from frequency modulation contains sidebands around the centre frequency. The faster the rate of modulation the more power is found over the sidebands. Time varying modulation produces complex and varying spectra. FM is ideal for bell-like sounds. Small changes in modulation frequency and amplitude can have a dramatic effect on the timbre of the resulting sound.

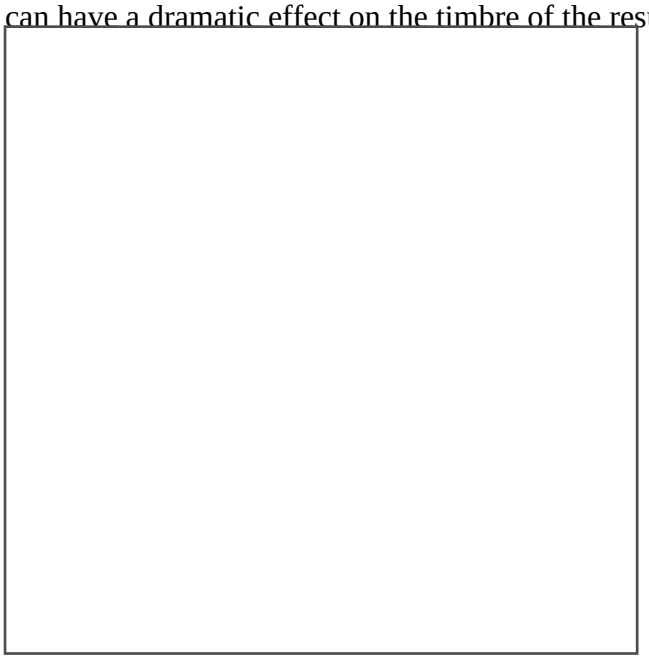

**Figure 1 An example FM instrument**

An example FM instrument is shown in figure 1. This instrument has one carrier oscillator and one modulating oscillator. $f_c$ is the output of the carrier if the modulating oscillator outputs zero. When modulation occurs, the frequency $f_c$ is modulated above and below the modulating oscillator output at a rate *d*. *AMP* is the amplitude of the output signal of the carrier oscillator. The maximum frequency produced is $f_c$ + d, and the minimum is $f_c$ - d. For example, with a value $f_c$ of 400, d of 50, $f_m$ of 100, and AMP=1000, then the carrier oscillator will be modulated between the values of 350 and 450 at a rate of 100 Hz, with a carrier oscillator output amplitude of 1000.

### 2.4 Amplitude And Ring Modulation

Amplitude modulation (AM) occurs when a sound's amplitude is modulated over time between 0 and a maximum value. The rate of modulation is determined by the frequency of the modulating oscillator. Ring modulation (RM) is similar to AM except amplitude (as well as the rate of oscillation) is determined directly by the modulating oscillator.

### 2.6 Waveshaping Synthesis

Waveshaping synthesis occurs when a waveform is taken and transformed. Usually a *transfer*

*function* is used to produce an output waveform based on the input waveform. Like FM synthesis, the outcome of subtle changes in parameters can not be often predicted especially if the waveform is altered by a non-linear processor.

### 2.7 Granular Synthesis

**Granular synthesis combines frequency and time domain synthesis** techniques. Several grains of sound are produced each with fixed frequencies and amplitudes. Many hundreds may be output in a single second producing a rich and varying texture. This texture can be difficult to control due to the large number of events involved, however granular synthesis can produce a variety of sounds including those resembling shattering of glass or torrents of water.

## 3 THE SOFTWARE ENVIRONMENT

Work has begun on the implementation of a distributed environment to allow the sharing synthesis techniques and eventually to allow real-time collaborative composition. The current prototype system uses SOAP based Web Services to allow access to a database of audio samples, audio processors and compositions.

### 3.1 Service Oriented Architectures

The current methodology in developing distributed systems is Service Oriented Architecture (SOA), building upon methodologies such as Object Oriented programming, Components and Distributed Object Request Brokers. Within a SOA, systems are composed of multiple individual services located and maintained on different heterogeneous machines administered by different organizations. The key in SOA is that the component services should be loosely coupled i.e. be well-defined, self-contained, and should not depend on the context or state of other services (6). To achieve this, a SOA should display the following properties:

- The services should implement a small set of simple, ubiquitous and well known interfaces which only encode generic semantics.
- The interfaces should deliver messages constrained by extensible schema for efficiency. This allows both services and consumers to work with well defined message structures, but allowing new versions of the services to be introduced without breaking existing systems
- The messages should be descriptive not instructive and the interfaces should not define system behaviour. This allows internals of a service can be viewed as a “black box”.
- Service Oriented Architectures must have mechanisms for the discovery of services matching the consumers requirements

There are a number of emergent technologies which can underpin SOA, namely REST WebServices; SOAP WebServices and GRID Services:

- Representational State Transfer (REST) works on the basic of “resources” which can be references by URIs (7). A REST web service is limited to using HTTP interfaces (GET to obtain a representation of the resource; DELETE to remove a representation of a resource; POST to update or create a representation of a resource; PUT to create a representation of a resource). REST messages are in XML, constrained by schema definitions in the XML

Schema language (http://www.w3.org/XML/Schema) or Relax NG (http://www.oasis-open.org/committees/tc_home.php?w-_abbrev=relax-ng)

- SOAP Web Services use messages encapsulated in a structure defined by the SOAP specification (http://www.w3.org/2000/xp/Group/). This adds additional information in the form of headers for message routing scenarios and mechanisms for reporting errors using faults (a style similar to exceptions in various programming languages). SOAP Web Services use Web Service Description Language (WSDL) to define both the structures (again using schema languages such as XML Schema) but also messaging semantics such as whether the message is initiated by the client or the server, and what messages can be used as a response to a particular message.
- GRIDServices are based on WebServices but provide additional semantics. In particular they add some object-oriented and REST concepts. The object-oriented concepts are the ability to inherit service definitions (portTypes in WSDL terminology) and add new messages using a multiple inheritance model and the ability to add properties (or service data elements) to WebServices. The REST concept introduced is that of creating a new representation of resource. In the GRIDServices model this uses a factory model whereby a new instance of a GRIDService can be created by its corresponding factory GRIDService. GRIDServices also offer an extensibility model whereby part of the structure of the message can be left undefined, but the allowed structures can be determined dynamically by querying the appropriate service data elements.

The current prototype is based upon SOAP based WebServices, although future versions may need to take advantage of GRIDService based technology for some of the more advanced features such as remote execution of compositions.

### 3.2 Web Service Definitions

The Web Service definitions define how the client communicates with the server using XML based messages. The prototype definitions are defined within WSDL which would allow clients and servers to be implemented on different platforms using different programming languages. The definition defines three different “PortTypes” (which is the WSDL term for a collection of functions). The division into different functional groups allows for a system in which different servers implement different groups, although our current prototype server includes implementations of all three. The defined PortTypes are:

#### *3.2.1 Processor Service PortType*

This defined two functions: **processAudio** and **getProcessorsDescriptions**. The **processAudio** function allows the client to send audio data to be processed at the server. It takes as its parameters an identifier, a collection audio sample and a collection of parameters and returns a single audio sample. The identifier determines what code will be used to process the audio using the various mechanisms described above. The collection of audio samples consists of MIME based 64 encoded binary of the actual wave data plus a name to identify that particular input to the code – typical names might be “input1”, “input2” etc. The collection of parameters also consists of a list of name, value pairs and allows fine control over the functioning of the server side sound synthesis code. The **getProcessorsDecriptions** returns a list of all the sound synthesis techniques available on the server. For each technique, it also returns the list of names for the input audio, the list of parameters including descriptions and type (e.g. integer, real or Boolean), and also the provenance of the sound synthesis technique using Dublin Core[] metadata

such as title, creator and description.

### *3.2.2 Audio Service PortType*

This defines functions for managing a server based database of audio samples. It implements the functions **getAudio** and **submitAudio** for adding new samples and retrieving existing samples. It also has the function **getAudioDescriptions** which returns a list of available samples and their provenance using Dublin Core metadata (title, description, creator, date etc.)

### *3.2.3 Composition Service PortType*

This defines functions for managing a server based database of compositions. A composition is represented as an XML document detailing the identifiers of the audio samples and processors used (i.e. the identifiers to use as a parameter for the **getAudio** and **processAudio** functions respectively) and the workflow i.e. how the audio and synthesis are composed to form the final audio. It implements the functions **getComposition** and **submitComposition** for adding new compositions and retrieving existing compositions. It also has the function **getCompositionsDescriptions** which returns a list of available compositions and their provenance using Dublin Core metadata (title, description, creator, date etc.)

## 3.3 Server Implementation

A prototype server has been developed in Java implementing all three port types, i.e. performing the roles of a audio repository, composition repository and a repository of various audio synthesis processors. This has been developed using OpenSource components such as Apache Tomcat (http://jakarta.apache.org) to provide the base HTTP functionality and the Apache Axis (http://ws.apache.org) to provide SOAP functionality.

The Audio and Composition repositories have currently been implemented a simple file stores with additional metadata store in an XML index file. The processor web service dynamically loads new processors by compiling and loading java classes stored in the processor directory. A web interface has been developed to allow remote uploading of new processors.

Helper java classes have been written to hide the Web Service complexity. To write a new synthesis processor, you have to implement a java class which has the method **processAudio**. The functions **getAudioOutput ()** is available to return a ByteArrayOutputStream to which to write the audio output, and functions **getAudioInput(String name)** and **getGlobalParameter(String name)** are provided to access the audio input and parameters. Use is made of the the new Java 1.5 metadata feature to add descriptive metadata to the source code about the creator and title of the processor, the names of the audio inputs and the names and types of the parameters which are returned by the **getProcesssorsDescriptions** WebService function. A simple processor which just adds samples together is show below (the lines beginning @ are the descriptive metadata using the new Java 1.5 features):

```
public class SampleAudioProcessor extends AbstractAudioProcessor {

  @Provenance(creator=”Dovey, Matthew”, title=”Simple Processor”)
  @AudioStreams({@AudioStreamDescription(name=”in1”, description=”first”),
                 @AudioStreamDescription(name=”in2”, description=”second”)})
  @Parameters({@ParameterDescription(name=”parm1”, description=””, type=”integer”)})

  public void processAudio() {
    byte array1[] = new byte[in1.available()];
    byte array2[] = new byte[in2.available()];

    this.getAudioInput("in1").read(array1);
    this.getAudioInput("in2").read(array2);

    for (int i = 0; i < array1.length && i < array2.length; i++) {
      this.getAudioOutput().write(array1[i] + array2[i] + intParameter(getGlobalParameter("parm1")));
    }
  }
}
```

A new processor is added to the server by dropping the source code in the processor directory with the extension .jps (either directly or via a web form). The file is automatically and dynamically detected and compiled by the server and the processor is then available for use.

### 3.4 Client Implementation

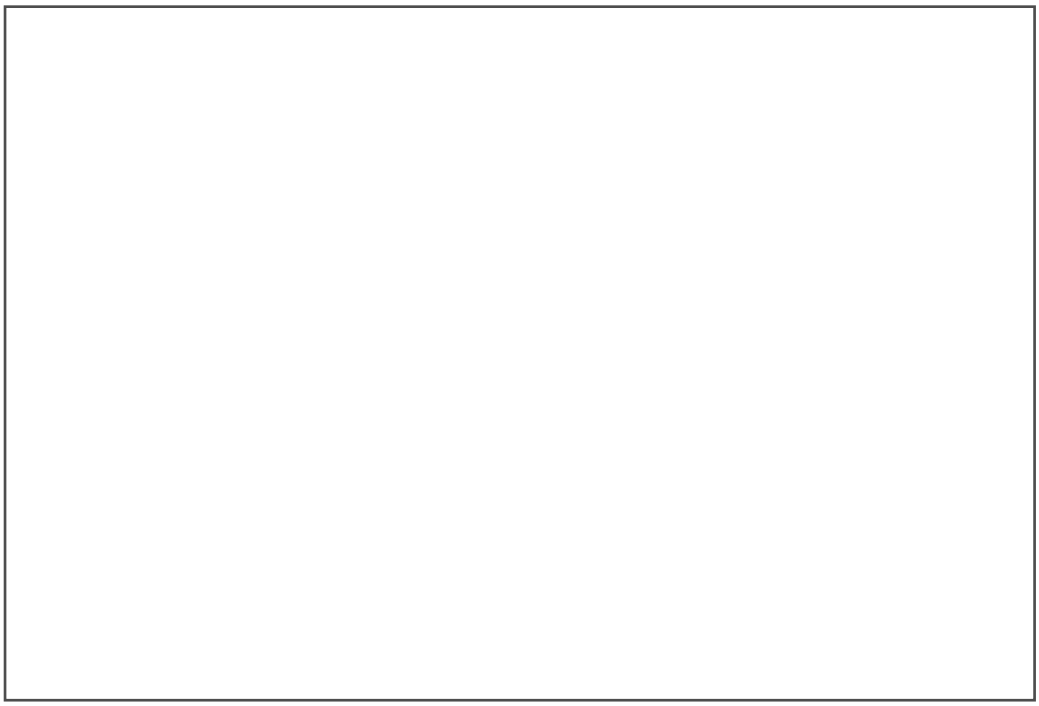

**Figure 2 The prototype client user interface**

A prototype client has been developed using Java Swing, and example of the user interface is show in figure 2. The client establishes a link to the server and uses the **getAudioDescriptions** and **getProcessorsDescriptions** WebServices to enumerate available

resources. This are listed in the Java JTree control on the left hand side. Audio samples and audio synthesis processors can be dragged and dropped onto the flowchart control on the right hand side which is implemented using the open source JGraph Swing control[1]. The various components can be then linked up to form a composition. Right-clicking on a processor brings up a list of the parameters that can be changed for that processor. The file menu allows the saving and loading of compositions either to the server (via **submitComposition** and **getComposition** WebServices) or to the local machine. The output of the composition can be sent to the speakers of the local machine, a local wave file or to the server (via the **submitAudio** WebService).

The current system is in an early prototype and still needs testing and feedback by users. However there are a number of areas which have already been identified as requiring further development

## 4 FUTURE RESEARCH

### 4.1 Searching

At present, a client must request the data for all the resources (audio, processors or composition) on the server in order to discover what is available. This is clearly not scalable as the number of resources grows. For this it would be better for the server to support a search interface so that audio processors, audio clips or compositions can be found matching a user's query. The use of the generic SRW WebService protocol[2] will be investigated for this purpose.

### 4.2 Multiple Servers

The architecture should be expandable to support multiple servers, so that the client might use processors, audio, etc. from numerous servers. For this to work there needs to be a naming convention so that a resource (audio sample or synthesis processor) and the server on which it is located can be determined from an identifier stored in a composition. Various systems such as WS-Addressing will be investigated for this purpose. There are also various architectures as to how a client discovers the various servers and there resources. In one case a client might send a get…Descriptions web service (or a search) to multiple servers simultaneously. An alternative might be for the servers to replicate metadata descriptions between themselves so that a client need only send a get…Descriptions or a search to a single server which will then response with information of the resource of all servers. It is likely that a hybrid approach will be implemented for greater flexibility.

### 4.3 Server execution of compositions

The current prototype is inefficient in the use of network traffic. In a typical composition, the client will retrieve a number of audio samples from the server using the **getAudio** WebService, then send those audio samples back over the network to the server for a synthesis technique to be applied via the **processAudio** WebService. The resultant audio will be transferred back to the client. As this may be an input to another audio processor the resultant audio may be passed back over the network to the server, and so on. A more efficient solution would be for the composition to be sent to the server and executed on the server. With multiple servers it will be necessary to calculate the most efficient workflow. GRID technologies will be investigated to provide this functionality.

### 4.4 Peer to Peer technologies

The prototype architecture is a client-server approach. A more flexible approach might be to use a peer to peer approach whereby a user can publish audio samples and synthesis processors from their local machine. Combining the client with the server would enable this, once a multiple server architecture is in place, although this may have some security implications to be investigated.

### 4.5 Instant Messaging Technologies

The system described is meant to encourage collaborative composition. This can only be achieved if the collaborative composers not only have access to shared resources but also to real time communication tools such as text based chatting, whiteboards, video/audio conferencing etc. Various technologies will be investigated for integration into the client.

## REFERENCES

(1) IMA (1988). MIDI 1.0 Detailed Specification, version 4.0. International MIDI Association.
(2) GIBSON I.S. (1997). Voice Analysis for Music Synthesis Systems. PhD Thesis (York, UK), 2.
(3) VERCOE, B.L. (1986). *The CSound Manual*. Cambridge Massachussets: Experimental Music Studio, Media Laboratory, MIT, Cambridge.
(4) FLANAGAN J.L. & GOLDEN, R.M. (1966). The Phase Vocoder. *Bell System Technical Journal*, **45**, 1493-1509.
(5) BAILEY N.J., PURVIS A., BOWLER I.W., MANNING P.D. (1993). Applications of the Phase Vocoder in the Control of Real-time Electronic Musical Instruments. *Interface*, **22**, 259 - 273.
(6) What is Service-Oriented Architecture.
http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html
(7) FIELDING, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Ph.D. Dissertation (University of California, Irvine), Chapter 5.

---

[1] http://www.jgraph.com
[2] http://www.loc.gov/z3950/agency/zing/srw/